IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANTS:	Nicholas J. Kelsey, et al.
SERIAL NO:	09/748,098
FILING DATE:	December 21, 2000
TITLE:	System and Method for Instruction Level Multithreading in a Embedded Processor Using Zero-Time Context Switching
EXAMINER:	David J. Huisman
ART UNIT:	2183
ATTY. DKT. NO.:	20880-05093

COMMISSIONER FOR PATENTS
P.O. BOX 1450
ALEXANDRIA, VA 22313-1450

APPEAL BRIEF

Real Parties in Interest

The subject application is owned by Ubicom, Inc. of Sunnyvale, California, Nicholas J. Kelsey of Mountain View, California and Christopher J. Walters of Palo Alto, CA.

Related Appeals and Interferences

There are no known related appeals or interferences that may directly affect, be directly affected by, or have a bearing on the Board's decision in the pending appeal.

Status of Claims

Claims 1-55 stand finally rejected. On August 23, 2007, Appellant appealed from the final rejection of claims 1-55. The claims on appeal are set forth in an appendix attached hereto.

Status of Amendments

Appellant has not amended the claims since the final rejection.

Summary of Claimed Subject Matter

The independent claims on appeal are claims 1, 17, 19 and 46. The claims comprise computer based systems (claims 1 and 17) and computer based methods (claims 19 and 46) for switching between program contexts. In one embodiment, a computer based system for switching between program contexts comprises a processor capable of having a first program thread and a second program thread includes thread selection hardware 802, 804 (e.g., FIGS. 5, 9; spec., page 27, line 4 to page 28, line 9). The system also includes a first set of data storage devices capable of storing a first thread state of the processor and a second set of data storage devices capable of storing a second thread state of said processor (e.g., FIG. 5; spec., page 5, lines 5-22; spec., page 14, line 21 to page 15, line 4). A hardware thread scheduler 802 identifies which of the first program thread and the second program thread that the processor executes (e.g., FIG. 9; spec., page 19, line 7 to page 22, line 18; spec., page 27, line 4 to page 28, line 9). The hardware thread scheduler 802 is configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread switching at a fixed time according to a predetermined fixed schedule (e.g., FIGS. 7A-7D; spec., page 19, line 6 to page 23, line 24). The predetermined fixed schedule specifies that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, where the first number of cycles is not equal to the second number of cycles (e.g., FIGS. 7A-7D; spec., page 19, line 6 to page 23 line 24). The thread selection hardware switches from the first thread state to the second thread state

between consecutive instructions cycles in response to identifying which program thread the processor executes (e.g., spec., page 16, line 15 to page 18, line 7). Further, the thread selection hardware in the processor switches from the first thread state to the second thread state between consecutive instruction cycles responsive to identifying which of the first thread or second thread is to be executed (e.g., spec., page 16, line 16 to page 18, line 7).

Similarly, the claimed method stores a first context of a processor in a first set of data storage devices, the first context corresponding to a first program thread and stores a second context of the processor in a second set of data storage devices, the second context corresponding to a second program thread (e.g., FIG. 5; spec., page 5, lines 5-22; spec., page 14, line 21 to page 15, line 4). The processor is then switched from the first thread state to the second thread state between the end of an execution cycle and before the beginning of a next consecutive execution cycle (i.e., between consecutive instruction cycles) at a fixed time according to a predetermined fixed execution schedule (e.g., FIGS. 7A-7D; spec., page 19, line 6 to page 23, line 24). The predetermined fixed schedule specifies that the first thread should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, where the first number of cycles is not equal to the second number of cycles (e.g., FIGS. 7A-7D; spec., page 19, line 6 to page 23 line 24; page 16, line 16 to page 18, line 7). The thread switching occurs by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector (e.g., FIG. 10, specification, page 28, lines 11-24; FIG. 11; specification page 30, lines 4-20).

Grounds of Rejection to be Reviewed on Appeal

Claims 2-18 and 19-28 have been rejected under 35 U.S.C. § 112, ¶ 1 as allegedly failing to comply with the enablement requirement. This rejection is improper because the claimed subject matter is described in the specification in such a way as to enable one skilled in the art to make and/or use the claimed invention. Specifically, independent claim 17 recites, "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes." Similarly, claim 19 recites "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle."

Claims 2-29 have been rejected under 35 U.S.C. §112, ¶ 2 as allegedly being indefinite for failing to particularly point out and distinctly claim the subject matter which is regarded as the invention. This rejection is improper because the claimed subject matter is described in the claims in such a way as to particularly point out and distinctly claim the subject matter regarded as the invention. Specifically, independent claim 17 recites, "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes." Similarly, claim 19 recites "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle."

Claims 1, 29-33, 42, 43 and 45-47 have been rejected under 35 U.S.C. §102(e) as allegedly being anticipated by U.S. Patent No. 6,076,157 to Borkenhagen et al. ("Borkenhagen"). This rejection is improper because Borkenhagen fails to disclose each and every element of the claimed invention. In particular, Borkenhagen fails to disclose "causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles," as claimed.

Claims 2-3, 13, 16-17 and 19-24 have been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable in view of U.S. Patent No. 6,542,991 to Joy et al. ("Joy") and U.S. Patent No. 6,493,741 to Emer et al. ("Emer"). This rejection is improper because the combination of Joy and Emer fails to disclose or suggest at least the claimed elements of "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle."

Claims 5-12, 18 and 25-28 have been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable in view of Joy and Emer in further view of U.S. Patent No. 6,085,215 to Ramakrishnan et al. ("Ramakrishnan"). This rejection is improper because the combination of Joy, Emer and Ramakrishnan fails to disclose or suggest at least the claimed elements of "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread

between the end of an execution cycle and before the beginning of a next consecutive instruction cycle."

Claim 14 has been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Joy and Emer in further view of Borkenhagen. This rejection is improper because the combination of Joy, Emer and Borkenhagen fails to disclose or suggest at least the claimed elements of "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle."

Claim 15 has been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Joy and Emer in further view of U.S. Patent No. 6,314,511 to Levy et al. ("Levy"). This rejection is improper because the combination of Joy, Emer and Levy fails to disclose or suggest at least the claimed elements of "said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle."

Claims 34-41 and 48-55 have been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Borkenhagen in view of Ramakrishnan. This rejection is improper because the combination of Borkenhagen and Ramakrishnan fails to disclose or suggest at least the claimed element of "causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every

first number of cycles and that the second thread should be allocated processing time every second number of cycles," as claimed.

Claim 44 has been rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over Borkenhagen in view of Levy. This rejection is improper because the combination of Borkenhagen and Levy fails to disclose or suggest at least the claimed element of "causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles," as claimed.

Argument

Rejections Under 35 U.S.C. § 112, ¶1, Claims 2-18 and 19-28

Regarding claim 17, the Examiner states "As is known, an instruction cycle (or clock cycle) is a period of time in which a clock oscillates from low to high. The cycle then repeats." *See* Final Office Action dated May 23, 2007, page 3, ¶ 6. However, independent claim 17 specifically recites "instruction cycles" and makes no mention of "clock cycles." Contrary to the Examiner's statement, as known in the art, an instruction cycle is distinct from a clock cycle. In particular, as known in the art, an instruction cycle describes the sequence of actions taken by a processor to execute an instruction from a particular thread, including fetching the instruction from memory, decoding and executing the instruction. *See*, e.g., Jack Ganssle and Michael Barr, *Embedded Systems Dictionary,* CMP Books, 2003, p. 138 ("instruction cycle: n. The time it takes to fetch and execute a single opcode."). Therefore, the claimed element of "between

consecutive instruction cycles" describes the interval between fetching a first instruction from a first thread and fetching a second instruction from the same first thread. Further, it is well-known in the art that multiple clock cycles may be required to fetch and execute an instruction.

Also in the Final Office Action dated May 23, 2007, the Examiner provided copies of references, including a Wikipedia reference and a slide from a course lecture, describing "the concept behind a clock cycle, i.e., an instruction cycle, and how there is no between cycles." *See* Final Office Action dated May 23, 2007, page 32, ¶ 77. Although Appellants object to the Examiner's use of such an unreliable source as Wikipedia as objective evidence of the state of the art at the time the application was filed, Appellants respectfully note that a search of Wikipedia for the clamed language of "instruction cycles" leads to an entry for "instruction cycle," which provides:

> The instruction cycle…can refer to either the time period during which one instruction is fetched from memory and executed when a computer receives a machine language instruction; or the sequence of actions that a CPU performs to execute each machine code instruction program.
>
> The name is quite literal. The instruction, along with any data to be worked on, must be fetched from main memory, and then executed by the CPU. (*See*, http://en.wikipedia.org/wiki/Instruction_cycle)

As searching for the specifically claimed element of "instruction cycles" leads to results different and unrelated to the term "clock cycles," Appellants respectfully submit that the Examiner was incorrect in equating the claimed language of "instruction cycles" to "clock cycles" to form the basis for the rejections under 35 U.S.C. § 112, ¶ 1

Additionally, the specification variously discloses thread switching between "instruction cycles" or between "machine instructions," and does not disclose thread switching between "clock cycles" nor associate "clock cycles" with "instruction cycles." *See,* e.g., specification

page 7, lines 16-26 and page 17, lines 1-24. Therefore, the "instruction cycles" recited in claim 17 refer to actions taken by a processor when executing an instruction from a first thread rather than the "period of time in which a clock executes from low to high" as alleged by the Examiner. Thus, there is no indication in the specification that "instruction cycles," as claimed, are similar to or interchangeable with, "clock cycles," as alleged by the Examiner.

Hence, the claimed "instruction cycles" describe actions taken by a processor when executing an instruction from a first thread rather than a period of time during which a clock oscillates from low to high. So, contrary to the Examiner's assertions, the system is not operating during an instruction cycle at any point in time, but is operating in an instruction cycle when an instruction from a first thread is fetched and executed and is not operating in an instruction cycle when no instruction from the first thread is being fetched and executed. Therefore, the specification contains "a written description of the invention…in such full, clear, concise and exact terms as to enable any person skilled in the art to which it pertains...to make and use the same." *See*, 35 U.S.C. §112 (emphasis added). Therefore, it is respectfully requested that the final rejections of independent claim 17 be withdrawn.

Pending claims 2-16 and 18 depend from independent claim 17, which is an enabled claim as discussed above. Accordingly, the arguments presented above are also applicable to claims 2-16, and it is respectfully requested that the final rejection of claims 2-16 and 18 also be withdrawn.

Regarding claim 19, the Examiner states "As is known, an execution cycle (or clock cycle) is a period of time in which a clock oscillates from low to high. The cycle then repeats." *See* Final Office Action dated May 23, 2007, page 4, ¶ 7. However, independent claim 19 specifically recites "execution cycle" and makes no mention of clock cycles. Contrary to the

Examiner's statement, as known in the art, an execution cycle is distinct from a clock cycle. As known in the art, an execution cycle is the period when a processor executes an instruction after the instruction has been fetched from memory and decoded. Hence, the execution cycle is part of the instruction cycle discussed above. *See*, e.g., Jack Ganssle and Michael Barr, *Embedded Systems Dictionary,* CMP Books, 2003, p. 138 ("instruction cycle: n. The time it takes to fetch and execute a single opcode."). Therefore, the claimed element of "between the end of an execution cycle and before the beginning of a next consecutive execution cycle" describes the interval between execution of an instruction from a first thread by the processor and execution of the next consecutive instruction from the same first thread by the processor. For example, the interval between execution of an instruction and execution of the next consecutive instruction in the same thread can be the time when the next instruction from a second thread is retrieved from memory and decoded or a time when the processor is not retrieving or executing instructions from memory. Also, it is well-known in the art that multiple clock cycles may be required to execute an instruction.

In the Final Office Action dated May 23, 2007, Examiner also provided copies of references, including a Wikipedia reference and slide from a course lecture, describing "the concept behind a clock cycle, i.e., an instruction cycle, and how there is no between cycles." *See* Final Office Action dated May 23, 2007, page 32, ¶ 77. Although Applicants object to the Examiner's use of such an unreliable source as Wikipedia as objective evidence of the state of the art at the time the application was filed, it is noted that a search of Wikipedia for the claimed language of "execution cycle" leads to an entry for "instruction cycle," which provides:

> Steps 3 and 4 of the Instruction Cycle are part of the Execute Cycle. These steps will change with each instruction. (*See*, http://en.wikipedia.org/wiki/Instruction_cycle)

As searching for the claimed element of "execution cycle," as specifically recited in claim 19 leads to results different and unrelated to the term "clock cycles," Appellants respectfully submit that the Examiner was incorrect in equating the claimed language of "execution cycle" to "clock cycle" to form the basis for the rejections under 35 U.S.C. § 112, ¶ 1.

Additionally, the specification variously discloses thread switching between "instruction cycles" or between "machine instructions," and does not disclose thread switching between "clock cycles" or associate "clock cycle" with "execution cycle." *See,* e.g., specification page 7, lines 16-26 and page 17, lines 1-24. Therefore, the "execution cycle" recited in claim 19 refers to actions taken by a processor to execute an instruction rather than the "period of time in which a clock executes from low to high" as alleged by the Examiner. Thus, there is no indication in the specification or elsewhere that the claimed "execution cycle" is similar to, or interchangeable with, "clock cycles," as alleged by the Examiner.

Hence, the claimed "execution cycle" describes actions taken by a processor when executing an instruction rather than a period of time during which a clock oscillates from low to high. So, contrary to the Examiner's assertions, the system is not operating during an execution cycle at any point in time, but is operating in an execution cycle when an instruction from a particular thread is being executed. Therefore, the specification contains "a written description of the invention…in such full, clear, concise and exact terms as to enable any person skilled in the art to which it pertains...to make and use the same." *See*, 35 U.S.C. §112 (emphasis added). Therefore, it is respectfully requested that the final rejections of independent claim 19 be withdrawn.

Pending claims 20-28 depend from independent claim 19, which is an enabled claim as discussed above. Accordingly, the arguments presented above are also applicable to claims 20-28, and it is respectfully requested that the final rejection of claims 20-28 also be withdrawn.

Rejections Under 35 U.S.C. § 112, ¶ 2, Claims 2-29

As discussed above regarding the rejections under 35 U.S.C. §112, "instruction cycles" and "execution cycle" as variously recited in independent claims 17 and 19 are both enabled by the specification and refer to terms that are known in the art. As such, the claimed elements of "between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle," are described in and throughout the specification in "such full, clear, concise and exact terms as to enable any person skilled in the art to which it pertains" to make and use the claimed invention. *See*, 35 U.S.C. § 112. Hence, because the claim language is clearly and concisely described in a way that would enable one skilled in the relevant art to make and use the invention, the claims particularly point out and distinctly claim the subject matter regarded as the invention. Therefore, it is respectfully requested that the final rejections of independent claims 17 and 19 be withdrawn. Similarly, as claims 2-16, 18 and 20-29 depend from independent claims that are definite, the arguments presented above are also applicable to claims 2-16, 18 and 20-29, and it is respectfully requested that the final rejection of claims 2-16, 18 and 20-29 also be withdrawn.

Rejections Under 35 U.S.C. § 102(e), Claims 1, 29-33, 42, 43 and 45-47

Representative claim 1 recites a computer based system for switching between program contexts comprising:

> a processor capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware;
>
> a first set of data storage devices capable of storing a first thread state of said processor;
>
> a second set of data storage devices capable of storing a second thread state of said processor; and
>
> a hardware thread schedule for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

Independent claim 46 recites a computer based method including similar claim elements. Hence, the claimed invention comprises a method and system for switching between program contexts. A processor capable of having at least two program threads in an execution pipeline stores a first processor thread state in a first set of data storage devices and stores a second thread state in a second set of data storage devices. A hardware thread scheduler is configurable to allocate processor processing time among at least the first and second program threads by switching between threads at a fixed time according to a predetermined fixed schedule. The predetermined fixed schedule specifies that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time and a thread scheduler that is configurable to allocate the processing time every second number of cycles where the first number of cycles is not equal the to second number of cycles. Switching threads

according to the predetermined fixed schedule beneficially provides a predictable execution time for the first thread and the second thread and allows different threads to be executed in a sequence specified by the predetermined fixed schedule. The predetermined fixed schedule is used to allocate computing time, for example quanta, which, for example, provides a guaranteed timely processing of hard-real-time processes. Further, the processor switches between threads in between the execution of two instructions, that is, in a zero-time manner, increasing computing speed.

In contrast, Borkenhagen merely discloses a multithreaded processor including a "time out register which forces a thread switch when execution of the active thread in the multi-threaded processor exceeds a programmable period of time." *See* Borkenhagen, Abstract. Therefore, the time-out register in Borkenhagen imposes an upper limit on thread execution time and forces a thread switch when the upper limit is reached. *See* Borkenhagen, col. 5, lines 45-55. Borkenhagen merely discloses that the time-out register holds a thread-switch time-out value and switches threads when the time-out value is reached. Borkenhagen, col. 14, lines 52-58; col. 15, lines 1-10. Unlike the claimed invention, which includes a "predetermined fixed schedule" specifying when a first and second thread should be allocated processing time, Borkenhagen associates a time-out value with each individual thread. The time-out value specifies a maximum processing time for a specific thread, but does not specify how processor time is allocated between a first thread and a second thread. Thus, the time out register in Borkenhagen does not specify that a "first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed.

Although Borkenhagen provides that different threads may have different time-out values, the time-out values and corresponding time-out registers do not specify allocation of processing time between a first thread and a second thread. The predetermined fixed schedule of the claimed invention specifies how processing time is allocated between two threads, allowing deterministic performance and predictable execution of the two threads. Hence, the claimed invention allocates processing between two threads using the predetermined fixed schedule, allowing processing time to be allocated among multiple threads before processing begins. The predetermined fixed schedule, as recited in claims 1 and 46, allows various threads to be allocated processor time at regular intervals specified in the predefined fixed schedule, guaranteeing deterministic performance for the scheduled threads.

In contrast, the time-out value and time-out register of Borkenhagen merely identify how long a specific thread receives processor resources. After the time-out value is exceeded, a thread switch is forced, but the time-out value and time-out register do not specify what thread is then allocated processor resources. After the time-out value has been reached, the time-out register forces a thread switch unless no other thread is ready to process instructions. If no other thread is ready to process instructions, the time-out register remains at zero until another thread is ready to process instructions at which point the thread ready to process instructions begins execution. So, rather than allocate processor time between various threads at regular intervals using a predefined fixed schedule, Borkenhagen merely switches to any available thread upon reaching the time-out value. *See* Borkenhagen, col. 15, lines 3-16. Rather than use a predefined fixed schedule to allocate processor time to different threads at specified intervals, Borkenhagen merely diverts processor time to any thread capable of processing instructions after the time-out value is reached. For example, in Borkenhagen, when a first thread reaches the time-out value,

processor time could be allocated to any of a second thread, third thread or fourth thread depending on which thread is ready to execute instructions. In contrast, the predetermined fixed schedule of the claimed invention provides that after executing the first thread for a predefined number of instruction cycles, processor time is allocated to a second thread specified by the predetermined fixed schedule, regardless of whether other threads are capable of processing instructions. Hence, the time-out value and time out register in Borkenhagen allow modification of processor resources allocated to a specific thread during execution, but are incapable of allocating processor resources among multiple threads prior to thread execution.

By failing to provide any reference disclosing "thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles," the Examiner has failed to establish *prima facie* anticipation and therefore these rejections are improper. Therefore, it is respectfully requested that the final rejections of independent claims 1 and 46 be withdrawn.

Pending claims 29-33, 42, 43, 45 and 47 variously depend from the above independent claims, and therefore incorporate the limitations of the above independent claims. Accordingly, the arguments presented above are also applicable to claims 29-33, 42, 43, 45 and 47.

Rejections Under 35 U.S.C. § 103(a), Claims 2, 3, 13, 16-17 and 19-24

Independent claim 17 recites:

> …said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles.

Similarly, independent claim 19 recites:

> …switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle.

Switching threads between consecutive instruction cycles or between consecutive execution cycles beneficially allows switching between one program context and another without incurring any time penalty. Switching from one thread to another between the end of an execution cycle and before the beginning of a next consecutive instruction cycle, or between consecutive instruction cycles, beneficially allows switching between thread contexts without loss of any execution cycles or instruction cycles.

Joy merely discloses that "the thread switch logic supports a fast thread switch with a very small, for example three cycles or less." Joy, col. 16, lines 61-62. Further, Joy discloses that the thread switch logic "generates the TID signal with a thread switch delay or overhead of one processor cycle. Joy, col. 16, lines 3-5. Hence, Joy discloses a thread-switching delay of one to three instruction cycles. Therefore, the thread switch logic in Joy executes a first thread during one instruction cycle, performs a thread switching operation during a second instruction cycle and executes a second thread during a third instruction cycle at the earliest. Hence, thread switching in Joy does not occur between consecutive instruction cycles, but occurs during an instruction cycle dedicated for instruction switching rather than instruction execution. In contrast, the claimed invention executes a first thread during one instruction cycle and executes a second thread during a second instruction cycle, having switched between the first thread and second thread between the instruction cycles.

Additionally, in the Final Office Action, the Examiner correctly states that Joy has "not taught that said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the

hardware thread scheduler identifying which of said program threads said pipelined processor executes." *See* Final Office Action dated May 23, 2007, ¶ 28, page 14. However, in the Final Office Action, the Examiner also cites sections of Joy describing thread switching every N cycles and that N may equal 1 so that thread switching occurs every cycle. *See* Final Office action dated May 23, 2007, ¶ 28, page 14. However, Joy merely discloses how often thread switching occurs, and does not address the time penalty associated with thread switches and does not disclose thread switching between consecutive instruction cycles rather than during an instruction cycle.

Emer fails to remedy the deficient disclosure of Joy. In the background section of Emer, a conventional multithreaded architecture capable of switching threads every cycle is disclosed. However, as discussed above regarding Joy, this disclosure merely describes how often thread switching occurs, and fails to disclose thread switching between consecutive instruction or execution cycles, as claimed. Specifically, Emer discloses:

> On any given cycle, a processor executes instructions from just one of the threads. On the next cycle, it switches to a different thread context and executes instructions from the new thread. (emphasis added)

Emer, col. 1, lines 54-58. Hence, Emer, like Joy, discloses switching from a first thread to a second thread during an instruction cycle, rather than between consecutive instruction or execution cycles. According to the disclosure of Emer, a processor executes a first thread during a first instruction cycle, switches from the first thread to a second thread during a second instruction cycle, and then executes the second thread during a second instruction cycle. Thus, Emer discloses thread switching during an instruction cycle rather than between consecutive instruction or execution cycles, as claimed.

As disclosed in Emer, multithreaded processors "better tolerate long-latency operations, effectively eliminating vertical waste." Emer, col. 1, lines 58-60. However, Emer characterizes effective elimination of vertical waste as thread switching that takes one or more instruction cycles to switch between threads but merely avoids instruction cycles that go completely unused because of long latency instructions. Emer, col. 1, lines 38-45. In light of this characterization, Emer discloses a thread switching system which improves processor efficiency by reducing processor stalls caused by long latency events. However, this thread switching system does not reduce or modify the necessary overhead to switch between threads. In addressing Emer, the Examiner asserts that "If a time penalty were incurred, then there would have to be some idle time in the processor which is not seen in Fig. 1(b)." *See*, Final Office Action dated May 23, 2007, page 22, ¶ 81. However, according to Emer's description of "effectively eliminating vertical waste," Figure 1(b) only depicts that a cycle was wasted if no instructions were executed. Hence, Emer does not account for a time penalty caused by using a cycle for both thread switching and instruction execution. As long as a single instruction from a thread is executed, Figure 1(b) of Emer indicates that the cycle was not wasted, even if the majority of the cycle was used for switching between threads rather than executing instructions. Further, Emer explicitly discloses that thread switching occurs during an instruction cycle, not between consecutive instruction cycles or execution cycles as recited in claims 17 and 19. Hence, contrary to the Examiner's assertion that there is no idle time in Emer, Emer discloses that an instruction cycle is partially used for switching between threads when a thread switch occurs. Emer, col. 1, lines 56-57.

Further, Examiner alleges that the application specification equates switching between instructions/cycles with switching without incurring a time penalty. *See* Final Office Action,

pages 33-34, ¶81. However, the portion of the application specification cited by the Examiner provides that:

> Zero-time context switching is the ability to switch between one program context and another without incurring any time penalty.

Specification, page 17, lines 9-13. Even in view of the Examiner's interpretation of this section, Emer does not disclose the claimed invention. Claims 17 and 19 variously recite "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle." Emer specifically discloses that thread switching occurs during an instruction cycle, not between consecutive instruction or execution cycles, as variously recited in claims 17 and 19. Emer explicitly discloses that conventional multithreaded processors "cannot removed [sic] horizontal waste." Emer, col. 1, lines 60-65. Emer further provides that "Horizontal waste occurs when some, but not all, of the issue slots in a cycle can be used." Emer, col. 1, lines 39-41. Hence, the multithreaded processors disclosed by Emer do incur a time penalty for thread switching, specifically horizontal waste.

Further, the Examiner incorrectly asserts that "there is no time penalty because the system is able to perform the maximum amount of work per cycle, i.e., execute one instruction form a given thread." *See* Final Office Action dated May 23, 2007, page 33, ¶ 81. Emer nowhere discloses that the disclosed multithreaded processor performs the maximum amount of work per cycle as the Examiner claims. In contrast, Emer describes thread switching where cycles do not go completely unused, so the system in Emer merely ensures that some minimum amount of instruction execution occurs during every cycle. *See* Emer, col. 1, lines 42-45.

Hence, Emer also fails to disclose "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle," as claimed.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claimed elements. See MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle," as claimed, the Examiner has failed to establish *prima facie* obviousness under MPEP §2143.03. Therefore, it is respectfully requested that the final rejections of independent claims 17 and 19 be withdrawn.

Pending claims 2-4, 13, 16 and 20-24 variously depend from the above independent claims, and therefore incorporate the limitations of the above independent claims. Accordingly, the arguments presented above are also applicable to claims 2-4, 13, 16 and 20-24.

Rejections Under 35 U.S.C. § 103(a), Claims 5-12, 18 and 25-28

Claims 5-12, 18 and 25-28 were rejected under 35 USC §103(a) as allegedly being unpatentable over Joy in view of Emer in further view of Ramakrishnan. Claims 5-12 and 18 depend from independent claim 17, so all arguments advanced above with respect to independent claim 17 are hereby incorporated so as to apply to claims 5-12 and 18. Claims 25-28 depend

from independent claim 19, so all arguments advanced above with respect to independent claim 19 are hereby incorporated so as to apply to claims 25-28.

Ramakrishnan fails to remedy the deficient disclosures of Joy and Emer. Rather, Ramakrishnan merely discloses a software scheduler using a round robin approach to thread scheduling using conventional context switching. *See* Ramakrishnan, col. 9, lines 9-10. The thread switching in Ramakrishnan, as disclosed in Joy and Emer, is conventional context switching which includes "an associated overhead in invoking the new thread." Ramakrishnan, col. 12, lines 61-62. *See also* Ramakrishnan, lines 6-7 ("avoid time consuming context switching.") Hence, Ramakrishnan also fails to disclose thread switching occurring between instruction cycles or between execution cycles, but merely discloses conventional thread switching occurring during an instruction cycle as variously recited in claims 17 and 19.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claim limitations. *See* MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles" and "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive instruction cycle," as claimed, the Examiner has failed to establish *prima facie* obviousness under MPEP §2143.03. Therefore, it is respectfully requested that the final rejections of claims 5-12, 18 and 25-28 be withdrawn.

Rejections Under 35 U.S.C. § 103(a), Claim 14

Claim 14 was rejected under 35 USC §103(a) as allegedly being unpatentable over Joy in view of Emer in further view of Borkenhagen. Claim14 depends from independent claim 17, so all arguments advanced above with respect to independent claim 17 are hereby incorporated so as to apply to claim 14.

Borkenhagen fails to remedy the deficient disclosures of Joy and Emer. Rather, Borkenhagen discloses a thread switching system which incurs conventional "latency and performance penalties associated with switching threads." Borkenhagen, col. 15, lines 37-38. In particular, Borkenhagen discloses:

> In the multithreaded processor in the preferred embodiment described herein, this latency includes the time required to complete execution of the current thread to a point where it can be interrupted and correctly restarted when it is next invoked, the time required to switch the thread-specific hardware facilities from the current thread's state to the new thread's state, and the time required to restart the new thread and begin its execution. (emphasis added)

Borkenhagen, col. 15, lines 38-46. Hence, the system in Borkenhagen conventionally switches between threads during an instruction cycle and incurs a temporal delay associated with thread-switching. Hence, Borkenhagen also fails to disclose "the pipelined processor switches from said first state to said second thread state between consecutive instruction cycles," so Borkenhagen also fails to remedy the deficient disclosures of Joy and Emer, both alone and in combination.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claim limitations. *See* MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive

instruction cycles," as claimed, the Examiner has failed to establish *prima facie* obviousness under MPEP §2143.03. Therefore, it is respectfully requested that the final rejection of claim 14 be withdrawn.

Rejections Under 35 U.S.C. § 103(a), Claim 15

Claim 15 was rejected under 35 USC §103(a) as allegedly being unpatentable over Joy in view of Emer in further view of Levy. Claim 15 depends from independent claim 17, so all arguments advanced above with respect to independent claim 17 are hereby incorporated so as to apply to claim 15.

Levy fails to remedy the deficient disclosures of Joy and Emer. Rather, Levy discloses a method for "freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of the time necessary for switching between threads or of when threads are switched. In contrast, Levy merely discloses a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. Thus, Levy also fails to disclose "the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles," as claimed, so Levy does not remedy the deficient disclosure of Joy and Emer, both alone and in combination.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claim limitations. *See* MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "said thread selection hardware in the pipelined processor switches from said first state to said second state between consecutive instruction cycles," as claimed, the Examiner has failed to establish *prima facie* obviousness

under MPEP §2143.03. Therefore, it is respectfully requested that the final rejection of claim 14 be withdrawn.

Rejections Under 35 U.S.C. § 103(a), Claims 34-41 and 48-55

Claims 34-41 and 48-55 have been rejected under 35 USC §103(a) as allegedly being unpatentable over Borkenhagen in view of Ramakrishnan. Claims 34-41 depend from independent claim 1, so all arguments advanced above with respect to independent claim 1 are hereby incorporated so as to apply to claims 34-41. Claims 48-55 depend from independent claim 46, so all arguments advanced above with respect to independent claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan fails to remedy the deficient disclosure of Borkenhagen. Rather, Ramakrishnan merely discloses a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. *See* Ramakrishnan, col. 9, lines 9-10. Thread selection in Ramakrishnan is determined by polling or by an interrupt system, rather than according to "a predetermined fixed schedule" for allocating processing time between a first thread and a second thread, as claimed. *See* Ramakrishnan, Abstract. The schedule in Ramakrishnan is not "a predetermined fixed schedule," but is modified based on thread polling or interrupts from threads. *See* Ramakrishnan, FIG. 4. Hence, the schedule is modified responsive to changes in the amount of work for different threads. Therefore, Ramakrishnan also fails to disclose "a predetermined fixed schedule" as variously recited in claims 1 and 46.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claim limitations. *See* MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "a predetermined fixed schedule," for

allocating thread processing time as claimed, the Examiner has failed to establish *prima facie* obviousness under MPEP §2143.03. Therefore, it is respectfully requested that the final rejections of claims 34-41 and 48-55 be withdrawn.

Rejections Under 35 U.S.C. § 103(a), Claim 44

Claim 44 was rejected under 35 USC §103(a) as allegedly being unpatentable over Borkenhagen in view of Levy. Claim 44 depends from independent claim 1, so all arguments advanced above with respect to independent claim 1 are hereby incorporated so as to apply to claim 33.

Levy fails to remedy the deficient disclosure of Borkenhagen. Rather, Levy discloses a method for "freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of "a predetermined fixed schedule" for allocating processing time between a first thread and a second thread, as claimed. In contrast, Levy merely discloses a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. As Levy fails to disclose "a predetermined fixed schedule" for allocating processing time between threads, Levy fails to remedy the deficient disclosure of Borkenhagen.

To establish a *prima facie* case of obviousness, the prior art references when combined must teach or suggest all the claim limitations. *See* MPEP §2143.03. On the contrary, by failing to provide any references that describe or suggest "a predetermined fixed schedule," for allocating thread processing time as claimed, the Examiner has failed to establish *prima facie*

obviousness under MPEP §2143.03. Therefore, it is respectfully requested that the final rejection of claim 44 be withdrawn.

Conclusion

For the foregoing reasons, the Examiner's rejection of claims 1-55 was erroneous, and reversal of the Examiner's decision is respectfully requested.

Respectfully submitted,
Nicholas J. Kelsey, et al.

Dated: January 12, 2008

By: /Brian G. Brannon/

Brian G. Brannon, Reg. No. 57,219
FENWICK & WEST LLP
801 California Street
Mountain View, CA 94041
Tel: (650) 335-7610
Fax: (650) 938-5200
bbrannon@fenwick.com

Appendix: Claims Involved in Appeal

1. A computer based system for switching between program contexts comprising:

a processor capable of having a first program thread and a second program thread in an execution pipeline having thread selection hardware;

a first set of data storage devices capable of storing a first thread state of said processor;

a second set of data storage devices capable of storing a second thread state of said processor; and

a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

2. The system of claim 17, wherein said first thread state is the thread state of the processor during the execution of the first program thread.

3. The system of claim 17, wherein said second thread state is the thread state of the processor during the execution of the second program thread.

4. The system of claim 17, wherein said processor switches between said first and second thread state by changing a state selection register.

5. The system of claim 17, wherein said hardware thread scheduler includes:

a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread;

an HRT scheduler for regularly scheduling an HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread in a predetermined time.

6. The system of claim 5, wherein said time quanta is at least-one instruction cycle.

7. The system of claim 5, wherein said thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said scheduled HRT is idle.

8. The system of claim 5, wherein said thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads.

9. The system of claim 8, wherein said thread scheduler regularly schedules NRT threads to be executed.

10. The system of claim 5, further comprising:

a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period;

a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period;

wherein said first fetch period is substantially shorter than said second fetch period.

11. The system of claim 10, wherein said first storage device for storing program instructions comprises a static RAM.

12. The system of claim 10, wherein said second storage device for storing program instructions comprises a flash memory.

13. The system of claim 17, wherein said processor is capable of restoring said second thread state of said processor during execution of said first program thread.

14. The system of claim 17, wherein said processor is capable of storing said second thread state of said processor during execution of said first program thread.

15. The system of claim 17, wherein said first set of data storage devices comprises registers shared by a plurality of threads.

16. The system of claim 17, wherein the execution schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule.

17. A computer based system for switching between program contexts comprising:

a pipelined processor capable of having a first program thread and a second program thread in an execution pipeline having thread selection hardware, the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage of the set of stages;

a first set of data storage devices capable of storing a first thread state of said pipelined processor;

a second set of data storage devices capable of storing a second thread state of said pipelined processor; and

a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule;

wherein said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes.

18. The system of claim 5, wherein said time quanta is exactly one instruction cycle.

19. A computer based method for switching between program contexts in a multithreading pipelined processor having a hardware thread selector and an execution pipeline, the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage of the set of stages, the method comprising:

storing a first context of said pipelined processor in a first set of data storage devices, the first context corresponding to a first program thread;

storing a second context of said pipelined processor in a second set of data storage devices, the second context corresponding to a second program thread;

switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a

next consecutive execution cycle by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector.

20. The method of claim 19, wherein the switching comprises changing a state selection register included in the hardware thread selector.

21. The method of claim 19, further comprising:

identifying which of the said program threads said processor executes according to an execution schedule.

22. The method of claim 21, further comprising:

allocating available processing time of the processor among at least the first and second threads according to the execution schedule.

23. The method of claim 22, wherein the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread.

24. The method of claim 23, wherein at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles.

25. The method of claim 21, wherein identifying further comprises identifying at least one hard-real-time (HRT) thread and at least one non-real-time (NRT) thread.

26. The method of claim 25, further comprising:

scheduling an HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread in a predetermined time.

27. The method of claim 25, further comprising:

scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle.

28. The method of claim 25, further comprising:

scheduling NRT threads in quanta not allocated for HRT threads.

29. The system of claim 1, wherein said thread selection hardware in the pipelined processor switches between said first and second thread state after the end of the execution of a first program instruction in the first thread and before the beginning of the execution of a second program instruction.

30. The system of claim 1, wherein said processor is an embedded pipelined processor.

31. The system of claim 1, wherein said first thread state is the thread state of the processor during the execution of the first program thread.

32. The system of claim 1, wherein said second thread state is the thread state of the processor during the execution of the second program thread.

33. The system of claim 1, wherein said processor switches between said first and second thread state by changing a state selection register.

34. The system of claim 1, wherein said hardware thread scheduler includes:

a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread;

a HRT scheduler for regularly scheduling an HRT thread according to the predetermined fixed schedule in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time.

35. The system of claim 34, wherein said time quanta is at least-one instruction cycle.

36. The system of claim 34, wherein said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said scheduled HRT thread is idle.

37. The system of claim 34, wherein said hardware thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads.

38. The system of claim 37, wherein said hardware thread scheduler regularly schedules NRT threads to be executed.

39. The system of claim 34, further comprising:

a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period;

a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period;

wherein said first fetch period is substantially shorter than said second fetch period.

40. The system of claim 39, wherein said first storage device for storing program instructions comprises a static RAM.

41. The system of claim 39, wherein said second storage device for storing program instructions comprises a flash memory.

42. The system of claim 1, wherein said processor is capable of restoring said second thread state of said processor during execution of said first program thread.

43. The system of claim 1, wherein said processor is capable of storing said second thread state of said processor during execution of said first program thread.

44. The system of claim 1, wherein said first set of data storage devices comprises registers shared by a plurality of threads.

45. The system of claim 1, wherein the predetermined fixed schedule is one of a fixed strict schedule or a semi-flexible strict schedule.

46. A computer based method for switching between program contexts in a multithreading pipelined processor having a hardware thread selector and an execution pipeline, the method comprising:

storing a first context of said processor in a first set of data storage devices comprising a first thread state corresponding to a first program thread;

storing a second context of said processor in a second set of data storage devices comprising a second thread state corresponding to a second program thread;

switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed

execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

47. The method of claim 46, wherein the switching comprises changing a state selection register included in the hardware thread selector.

48. The method of claim 46, further comprising:

identifying which of the program threads said processor executes according to a hard-real-time (HRT) execution schedule.

49. The method of claim 48, further comprising:

allocating available processing time of the processor among at least the first and second threads according to the predetermined fixed execution schedule.

50. The method of claim 49, wherein the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread.

51. The method of claim 50, wherein at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles.

52. The method of claim 48, wherein identifying further comprises identifying at least one hard-real-time (HRT) thread and at least one non-real-time (NRT) thread.

53. The method of claim 52, further comprising:

scheduling a HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time.

54. The method of claim 52, further comprising:

scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle.

55. The method of claim 52, further comprising:

scheduling NRT threads in quanta not allocated for HRT threads.

Evidence Appendix

None.

Related Proceedings Appendix

None.